

Appl. No. 10/692,115  
Appeal Brief dated 02/25/2008  
Response to Office Action of 09/25/2007

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

In re: Application of:	:	
Cary Lee Bates	:	
	:	Before the Examiner:
Serial No: 10/692,115	:	Zheng Wei
	:	
Filed: 11/23/2003	:	Group Art Unit: 2192
	:	
Title: TRACKING AND MAINTAINING	:	Confirmation No.: 6581
RELATED AND DERIVATIVE CODE	:	

APPELLANTS' BRIEF UNDER 37 CFR §41.37

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

Sir:

This is an Appeal to a Final rejection dated September 25, 2007 of the claims in the Application. This Brief is submitted pursuant to a Notice of Appeal filed on December 25, 2007 in accordance with 37 CFR §41.31.

CA920020065US1

BRIEF FOR APPLICANTS – APPELLANTS

(i)

Real Party in Interest

The real Party in interest is International Business Machines Corporation (IBM), the assignee.

(ii)

Related Appeals and Interferences

There are no other appeals or interferences known to appellants, appellants' representative or assignee, which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(iii)

Status of Claims

Claims 1 – 23 were originally filed in the Application and rejected in a Final Office Action. In that Final Action, independent Claim 10 was rejected under 35 USC §112, second paragraph and independent Claim 18 under 35 USC §101. Due to these rejections, independent Claims 10 and 18 are not being appealed. Further, Claim 19, which depends on independent Claim 18 is not being appealed either. Thus, the claims being appealed are Claims 1 – 9, 11 – 17 and 20 – 23.

(iv)

Status of Amendment

An "Amendment after Final" was not filed.

(v)

Summary of Claimed Subject Matter

The invention, as claimed in Claim 1, provides an algorithm to improve efficiency of editing source code. The algorithm comprises: recognizing that a  
CA920020065US1

source code has been edited (lines 8 and 9 of the last paragraph on page 10 and step 330 in Fig. 3); identifying a program construct having the edited source code (lines 9 – 11 of the last paragraph on page 10 and step 350 in Fig. 3); constructing a construct list of at least one other construct having derived and/or related code to the program construct (lines 1 – 3 of the second paragraph on page 12, third line of full paragraph on page 14); comparing the at least one other construct with the program construct having the edited source code (lines 1 – 3 of the second paragraph on page 12 and step 650 in Fig. 6); and if, in response to comparing the at least one other construct with the program construct, a commonality between the at least one other construct and the program construct is found to be equal to or beyond a threshold of similarity, then notifying a user responsible for the at least one other construct that the source code of the program construct has been edited (lines 14 – 21 of the second paragraph on page 12 and steps 665 and 680 in Fig. 6).

The invention, as claimed in Claim 11, provides a method of determining if two or more constructs in a repository of source code in an integrated development environment are related to each other. The method comprises the steps of: identifying a first construct (lines 10 and 11 of the last paragraph on page 10 and step 360 in Fig. 3); parsing N tokens of the first construct, N being a positive integer (lines 2 and 3 of the second paragraph on page 11 and step 420 in Fig. 4, line 1 of the first paragraph on page 13, lines 2 and 3 of the second paragraph on page 13); identifying a plurality of other constructs in the repository (lines 1 – 3 of the second paragraph on page 12); parsing M tokens of each one of the other constructs, where  $M = N$  (first and second paragraphs on page 13); comparing the M tokens of each one of the other constructs with the N tokens of the first construct (first and second paragraphs on page 13); determining a weight for each one of the N and M tokens based on name, type, and/or representation (the last three lines of the last paragraph on page 13); summing the weights of the N and M tokens (columns A and B on page 14); determining whether the sum of the weights of the M tokens meets or exceeds a threshold of

CA920020065US1

similarity, the threshold of similarity being based on a percentage of the sum of the weights of the M tokens to the sum of the weights of the N tokens (first and second paragraph on page 13); and identifying each construct whose sum of the weights of the M tokens meets or exceeds the threshold of similarity as being related to the first construct (lines 4 – 9 of the second paragraph on page 10 and first and second paragraphs on page 13).

he invention, as claimed in Claim 16 provides an integrated development environment. The integrated development environment comprises: a repository of source code into which programs in the integrated development environment are stored (repository 44 in Fig. 2, lines 2 – 7 of the last paragraph on page 10); a constructor to determine, when executed by a processor, whether a construct in an edited program has been edited (constructor 50 in Fig. 2, lines 9 and 10 of the last paragraph on page 10); construct list within the repository into which all constructs that are derived and/or related to the edited construct are listed (last five lines of the last paragraph on page 10); a parser to parse the edited construct and the derived and/or related constructs, when executed by a processor (line 4 of the second paragraph on page 10, lines 3 and 4 of the second paragraph on page 11); a matchmaker to determine, when executed by a processor, all the constructs that are derived and/or related to the edited construct and to enter all constructs determined to be derived and/or related to the edited construct in the construct list (matchmaker 60 in Fig. 2, last three lines of the last paragraph on page 10); and an announcer to notify, when executed by a processor, any of a plurality of programmers accessing the integrated development environment that the edited construct has been edited and the constructs that are derived and/or related to the edited construct (announcer 70 in Fig. 2, last line of the last paragraph on page 10 and first two lines of the first paragraph on page 11).

The invention, as claimed in Claim 20, provides an article of manufacture. The article of manufacture comprises a data storage medium tangibly embodying a program of machine readable instructions executable by an electronic

CA920020065US1

processing apparatus to perform method steps for operating an electronic processing apparatus. The method steps comprises the steps of: determining that a source code has been edited in an environment of computer program development (lines 8 and 9 of the last paragraph on page 10 and step 330 in Fig. 3); determining if the edited source code is within a construct of a size of a particular range (line 4 to last line of the second paragraph on page 11 and the steps 430, 435, 440, 450 and 460 in Fig. 4); parsing the construct having the edited source code if the edited source is within a construct of the size of the particular range (lines 2 and 3 of the second paragraph on page 11 and steps 410 and 420 in Fig. 4); finding and parsing other constructs in the environment having a size within the particular range (line 4 of the second paragraph on page 11 and step 430 in Fig. 4); creating a construct list of other constructs in the environment having a size within the particular range (third paragraph on page 11 and steps 410 and the steps in Fig. 5); comparing tokens of the construct having the edited source code with tokens of the other constructs in the construct list (lines 1 – 11 of the second paragraph on page 12 and steps 610, 620, 630, 640, 650 and 660 in Fig. 6); and determining that the construct having the edited source code is related to any one of the constructs in the construct list if the tokens of any one of the constructs in the construct list equal the tokens of the construct having the edited source code (first paragraph on page 13).

(vi)

Grounds of Rejection to be Reviewed on Appeal

**Whether it was proper to reject Claims 1 – 8 under 35 USC §102(e) as being anticipated by Fogel et al.**

**Whether it was proper to reject Claims 11 – 15 and 20 - 23 under 35 USC §102(b) as being anticipated by Bloom**

**Whether it was proper to reject Claims 16 and 17 under 35 USC §103(a) as  
being unpatentable over Fogel et al. in view of CvsIn**

(vii)

Arguments

**Whether it was proper to reject Claims 1 – 8 under 35 USC §102(e) as being  
anticipated by Fogel et al.**

**Claim 1**

Fogel et al. disclose a manual that describes how to use and administer CVS (Concurrent Versions System) for collaboration and version control. In the manual, it is disclosed that CVS: (1) enables a plurality of developers to edit a program simultaneously, (2) assumes the burden of integrating all the changes in the program, and (3) keeps track of any conflicts by using the copy-modify-merge model. For example, developer A and developer B may each request a working copy of a program from CVS. Both developers may edit freely their working copy. When a first developer (e.g., developer A) is done, the first developer may try to commit the changes made to the program into CVS. CVS, at this point, will incorporate the changes into the program. When the second developer (e.g., developer B) is done, the second developer may also try to commit the changes made to the program into CVS. In this case, before the changes are incorporated into the program, CVS will compare the changed copy of developer A with the changed copy of developer B. If there is a conflict (e.g., if both developers changed the same lines of code), CVS will flag the changes in the copy of developer B. At this point, it is up to developer B, with or without the help of developer A, to resolve the conflict. When the conflict is resolved, developer B may then retry to commit the changes to the program by again submitting the edited copy of the program to CVS. If there continues to be a conflict, the working copy of developer B will be rejected and the conflicting code flagged as before. If, on the other hand, there is no longer a conflict, the changes made by

CA920020065US1

developer B will be incorporated into the program. Therefore, the program will now include the changes from developers A and B.

Thus, CVS compares copies of files to determine whether (1) there are changes in the copies of the files and if at least two copies of the files have changes therein then (2) whether the changes in one copy conflict with the changes in the other copy. If so, then (3) the conflict will be flagged and (4) a user notified of the conflict.

By contrast, the claimed invention compares a changed construct in a file (that is already known to have had changes therein) with constructs in the same file and/or other files that have not been changed for instance. The comparison is done to determine whether there are other constructs in the file or constructs in the other files that were derived from the changed construct. If so, then a user responsible for the other files or constructs may be notified in order for the user to accordingly make changes in those constructs derived from the changed construct.

Note that a construct is a small piece of source code which makes up, for instance, a CASE statement or an IF statement or a DO loop in a program (see last paragraph on page 2 of Appellants' Specification).

Thus, Fogel et al. do not teach, show or so much as suggest the steps of (1) identifying a program construct having the edited source code; (2) constructing a construct list of at least one other construct having derived and/or related code to the program construct; (3) comparing the at least one other construct with the program construct having the edited source code; and (4) if, in response to comparing the at least one other construct with the program construct, a commonality between the at least one other construct and the program construct is found to be equal to or beyond a threshold of similarity, then notifying a user responsible for the at least one other construct that the source code of the program construct has been edited as claimed in claim 1.

In the RESPONSE TO ARGUMENTS section in the Final Office Action, the Examiner asserted that since the term **program construct** is not defined in CA920020065US1

the claims then it can reasonably be interpreted as a whole file of source code or a character of the source code. In the case of Claim 1, the Examiner interpreted the size of the **construct** as being the same size as a file size. The Examiner then concluded that the reference teaches the claimed invention. Appellant respectfully disagrees.

First of all, claim elements are given their broadest interpretation, not in a vacuum, but rather consistent with the SPECIFICATION of the Application. As pointed out in the Response to the 1<sup>st</sup> Office Action, the definition of “program construct” does not entail an entire program in the SPECIFICATION.

Secondly, let us suppose that it is indeed reasonable to interpret the term **program construct** as a whole file of source code, as the Examiner asserted, Fogel et al. would then disclose: (1) recognizing that a source code has been edited; identifying a FILE having the edited source code.

But note that Fogel et al. do not teach anywhere in their disclosure the step of constructing a file list of at least one other file having derived and/or related code to the file. Therefore, even if it were reasonable to interpret program construct as a whole file of source code, Fogel et al. would not teach the step of ***constructing a construct list of at least one other construct having derived and/or related code to the program construct.***

Further, Fogel would not teach the step of ***comparing the at least one other construct with the program construct having the edited source code*** and neither would Fogel et al. teach the step of ***if, in response to comparing the at least one other construct with the program construct, a commonality between the at least one other construct and the program construct is found to be equal to or beyond a threshold of similarity, then notifying a user responsible for the at least one other construct that the source code of the program construct has been edited.***

It should nonetheless be emphasized that in the claimed invention a user is notified only ***if a commonality between the program construct and the at least one other construct is equal to or is beyond a threshold of similarity.***



Fogel et al. do not teach anywhere in their disclosure such limitations.

It is a well settled law that in considering a Section 102 rejection, all the elements of the claimed invention must be disclosed in a single item of prior art in the form literally defined in the claim. *Jamesbury Corp. v. Litton Indus. Products*, 756 F.2d 1556, 225 USPQ 253 (Fed. Cir. 1985); *Atlas Powder Co. v. Dupont*, 750 F.2d 1569, 224 USPQ 409 (Fed. Cir. 1984); *American Hospital Supply v. Travenol Labs.*, 745 F.2d 1, 223 USPQ 577 (Fed. Cir. 1984).

As shown above, Fogel et al. do not teach all the elements of the claim in the form literally defined in the claim. Thus, Appellant submits that Claim 1 is not anticipated by Fogel et al.

### **Claim 2**

Claim 2 includes the limitations “wherein identifying the program construct further comprises parsing tokens of the edited source code.”

The Examiner asserted that Fogel et al. teach these limitations on page 21 – 23 under the “Finding Out What You (And Others Did) section. Appellant disagrees.

In that section, Fogel et al. disclose that CVS will compare a copy of a file in repository with a working copy of the file to determine whether there are changes made to either the copy in the repository or the working copy.

However, Fogel et al. do not disclose any teaching that includes **parsing tokens** of the edited source code.

Consequently, Appellant submits that Claim 2 is patentable over the applied reference.

### **Claim 3**

Claim 3 includes the limitations “wherein constructing a construct list further comprises determining that the at least one other construct is of at least a threshold size for placement in the construct list.”

The Examiner asserted that Fogel et al. disclose the claimed limitations on page 18, "Checking Out A Working Copy" section. Appellant disagrees.

In that section, Fogel et al. disclose the different commands that may be used to check out a file.

However, Fogel et al. do not disclose the limitations "wherein constructing a construct list further comprises determining that the at least one other construct is of at least a threshold size for placement in the construct list."

Hence, Appellant submits that Claim 3 is also patentable over the applied reference.

#### **Claim 4**

Claim 4 includes the limitations "further comprising parsing a sequence of tokens from each of a plurality of constructs of the at least threshold size."

The Examiner asserted that Fogel et al. teach these limitations on page 21 "Finding Out What You (And Others Did)" section. Appellant disagrees.

As mentioned above, in that section, Fogel et al. disclose that CVS will compare a copy of a file in repository with a working copy of the file to determine whether there are changes made to either the copy in the repository or the working copy.

However, Fogel et al. do not disclose any teaching that includes **parsing tokens**, much less parsing a sequence of tokens from each of a plurality of constructs of the at least threshold size.

Consequently, Appellant submits that Claim 4 is patentable over the applied reference.

#### **Claim 5**

Claim 5 includes the limitations "wherein comparing the at least one other construct with the program construct further comprises comparing the parsed tokens of the edited source code with parsed tokens of each of a plurality of constructs in the construct list."

CA920020065US1

The Examiner asserted that Fogel et al. teach these limitations on page 21 “Finding Out What You (And Others Did)” section. Appellant disagrees.

Again, in that section, Fogel et al. disclose that CVS will compare a copy of a file in repository with a working copy of the file to determine whether there are changes made to either the copy in the repository or the working copy.

However, Fogel et al. do not disclose any teaching that includes **parsing tokens**, much less comparing the parsed tokens of the edited source code with parsed tokens of each of a plurality of constructs in the construct list.

Consequently, Appellant submits that Claim 5 is patentable over the applied reference.

#### **Claim 6**

Claim 6 includes the limitations “wherein comparing the parsed tokens further comprises weighting the compared tokens to establish a degree of similarity.”

The Examiner asserted that Fogel et al. teach these limitations on page 21 “Finding Out What You (And Others Did)” section. Appellant disagrees.

Once more, in that section, Fogel et al. disclose that CVS will compare a copy of a file in repository with a working copy of the file to determine whether there are changes made to either the copy in the repository or the working copy.

However, Fogel et al. do not disclose any teaching that includes **parsing tokens**, much less the step of comparing the parsed tokens that comprises weighting the compared tokens to establish a degree of similarity.

Consequently, Appellant submits that Claim 6 is patentable over the applied reference.

#### **Claim 7**

Claim 7 includes the limitations “further comprising summing the weights of the compared tokens to determine if the sum is equal to or beyond the threshold of similarity.”

CA920020065US1

The Examiner asserted that Fogel et al. teach these limitations on page 27 “see all the changes at once” section. Appellant disagrees.

In that section, Fogel et al. disclose that when a command to see all the differences between a working copy of a file and a copy of the file in a repository is issued, all the differences between the two copies will be provided.

However, Fogel et al. do not disclose any teaching that includes summing weights of compared tokens to determine if the sum is equal to or beyond the threshold of similarity.

Consequently, Appellant submits that Claim 7 is patentable over the applied reference.

### **Claim 8**

Claim 8 includes the limitations “further comprising storing the construct list.”

The Examiner asserted that Fogel et al. teach these limitations on page 99 “The Checkoutlist File” section. Appellant disagrees.

Firstly, as mentioned above, Fogel et al. do not teach the step of making a list of constructs.

Thus, Fogel et al. cannot disclose the step of storing the construct list.

Secondly, in the section cited by the Examiner, Fogel et al. disclose that when a working copy of CVSROOT is committed to CVS, CVS automatically updates any changed files in the repository.

But, nowhere in that section is there a teaching of storing a construct list.

Hence, Appellant submits that Claim 8 is patentable over the applied reference.

### **Whether it was proper to reject Claims 11 – 15 and 20 - 23 under 35 USC §102(b) as being anticipated by Bloom**

### **Claim 11**

CA920020065US1

Bloom purports to teach a source code comparator computer program. According to the teachings of Bloom, the comparator computer program compares an original version of a source program to a (possibly) modified version of the program to identify the difference between the two. The program compares each line in the two versions of the source program until a non-comparison (i.e., a difference) is detected. That point of difference is marked. Then the program searches for two consecutive lines in the two versions of the source program that are the same. This marks the end of the difference between the original source program and the modified source program. This test will henceforth be referred to as the forward comparison test.

According to Bloom, the two consecutive lines found may be misleading. This can occur when lines of the original source program, which are deleted in the modified source program, appeared later in the two programs (see col. 4, lines 39 – 43). Therefore, Bloom advocates finding an identical symbolic address in both the original source program and the modified source program that is downstream from the marked point of difference. This identical symbolic address represents a common point where the particular coding format that is presently being compared ends and another format of the source listing begins.

From that symbolic address, lines of the original source program are compared to lines of the modified source program going in a backward direction (i.e., from the symbolic address toward the marked point of difference). This test will be referred to as the backward comparison test.

In the backward comparison test, the lines of the original source program should be different from the ones of the modified program since in this case the comparison is being done over the part of the (modified) program that has been modified. When a line of the source program is the same as that of the modified program is encountered, it signifies that the last line of code that was not different in the first comparison test (i.e., at the marked point of difference above) is found.

Generally, the beginning and end of each modified piece of code should be the same in both the forward comparison test and the backward comparison

test. The only time it is not different is when an erroneous assumption was made (e.g., when lines of the original source program, which are deleted in the modified source program, appeared later in the two programs are taken as two consecutive lines that are the same in both versions of the source program). In such a case, Bloom advocates choosing the test that yields the fewest number of modified lines (i.e., the smallest area of change).

The Examiner in the Response to Argument section of the Final Office Action asserted that Bloom discloses "Select Test Producing Smallest Area of Change." In order to select smallest area of change, the Examiner reasoned, weights in doing the comparison have to be used. Therefore, Bloom teaches the limitations of Claim 11. Appellant disagrees.

As mentioned above, selecting the test that produces the smallest area of change does not, impliedly or otherwise, teach the use of weights. Rather, Bloom simply advocates using the test that yields the fewest number of modified lines.

Further, it should be noted that in addition to the weights, Bloom does not teach the use of tokens, threshold of similarity and constructs. More specifically, Bloom does not teach the steps of (1) identifying a first construct; (2) parsing N tokens of the first construct, N being a positive integer; (3) identifying a plurality of other constructs in the repository; (4) parsing M tokens of each one of the other constructs, where  $M = N$ ; (5) comparing the M tokens of each one of the other constructs with the N tokens of the first construct; (6) determining a weight for each one of the N and M tokens based on name, type, and/or representation; (7) summing the weights of the N and M tokens; (8) determining whether the sum of the weights of the M tokens meets or exceeds a threshold of similarity, the threshold of similarity being based on a percentage of the sum of the weights of the M tokens to the sum of the weights of the N tokens; and (9) identifying each construct whose sum of the weights of the M tokens meets or exceeds the threshold of similarity as being related to the first construct as in the claimed invention.

Therefore, Appellant submits that Claim 11 is not anticipated by Bloom.

### **Claim 12**

Claim 12 includes the limitations “wherein the step of identifying the first construct further comprises the step of identifying whether a source code within which resides the first construct has been edited.”

The Examiner asserted that Bloom teaches the claimed limitations in Fig. 1 the step “Determine if Change Is Deletion, Addition or Modification Line-by-Line” and related text. Appellant disagrees.

As mentioned above, Bloom does not teach the step of identifying a first construct. Therefore, Bloom cannot teach the step of identifying whether a source code within which resides the first construct has been edited.

Hence, Claim 12 is not anticipated by Bloom.

### **Claim 13**

Claim 13 includes the limitations “further comprising storing a pointer to each construct identified as being related to the first construct in a construct list of related constructs.”

The Examiner asserted that Bloom teaches the claimed limitations in Fig. 2C step 66 “Set Pointer Based on Comparison Having Least Change” and related text.

In the text explaining step 66, Bloom discloses:

The next block in the flow diagram, block 66, sets the pointer based on the comparison having the least change. It is in this portion of the comparison flow that positively identifies the area of the change. The pointers are set to identify the line where the first noncomparison was detected and the line where the noncomparison ends. By performing the forward and the reverse comparisons all changes are located.

Thus, Bloom does not teach the step of storing pointers much less teaching ***storing a pointer to each construct identified as being related to the first construct in a construct list of related constructs.***

Thus, Claim 13 is patentable over the applied reference.

#### **Claim 14**

Claim 14 includes the limitations “further comprising the step of identifying users responsible for each of the constructs in the construct list.”

The Examiner asserted that Bloom teaches the claimed limitations in Fig. 2A, step 12 “Read in Options Selected;” “setup Compare Parameters”, step 20 “Initialize Search Flags” and related text.

But note that the Examiner does not so much insinuate that Bloom teaches identifying users responsible for each construct in construct list.

Hence, Appellants submit that Claim 14 is not anticipated by Bloom.

#### **Claim 15**

Claim 15 includes the limitations “further comprising the step of offering notification to each user responsible for each one of the constructs in the construct list.”

The Examiner asserted that Bloom teaches the claimed limitations in Fig. 1, the steps “Determine if Change Is Deletion, Addition or Modification Line-by-Line”, “Print Change” and related text.

But as in the case of Claim 14 above, the Examiner does not so much insinuate that Bloom teaches the step of offering notification to each user responsible for each one of the constructs in the construct list.

Hence, Appellant submits that Claim 15 is not anticipated by Bloom.

#### **Claim 20**



As mentioned above (see Claim 11 above), Bloom purports to teach a source code comparator computer program for comparing two versions of a source program to identify differences therebetween.

However, Bloom does not teach (1) determining that a source code has been edited in an environment of computer program development; (2) determining if the edited source code is within a construct of a size of a particular range; (3) parsing the construct having the edited source code if the edited source is within a construct of the size of the particular range; (4) finding and parsing other constructs in the environment having a size within the particular range; (5) creating a construct list of other constructs in the environment having a size within the particular range; (6) comparing tokens of the construct having the edited source code with tokens of the other constructs in the construct list; and (7) determining that the construct having the edited source code is related to any one of the constructs in the construct list if the tokens of any one of the constructs in the construct list equal the tokens of the construct having the edited source code.

Hence, Claim 20 is patentable over the applied reference.

#### **Claim 21**

Claim 21 includes the limitations “further comprising weighting the compared tokens based on value, type, and/or representation.”

As mentioned earlier, Bloom does not teach weights and tokens. Further, Bloom does teach comparing tokens based on value, type, representation.

Thus, Claim 21 is not anticipated by Bloom.

#### **Claim 22**

Claim 22 includes the limitations “wherein notifying a user responsible for of any one of the constructs in the construct list that is determined to be related to the construct having the edited source code that the source code of the construct has been edited.”

CA920020065US1

As mentioned earlier, Bloom does not teach notifying users responsible for program codes.

Thus, Claim 22 is not anticipated by Bloom.

### **Claim 23**

Claim 23 includes the limitations “further comprising at least one construct list of related and/or derived constructs within the integrated development environment.”

Bloom does not teach constructs and construct lists.

Thus, Claim 23 is not anticipated by Bloom.

### **Whether it was proper to reject Claims 16 and 17 under 35 USC §103(a) as being unpatentable over Fogel et al. in view of CvsIn**

### **Claim 16**

The Examiner stated that Claim 16 is an integrated development environment (IDE) system claim version of method Claim 1. Therefore, using CvsIn to show that IDE is known in the art, the Examiner rejected the claim under 103 by combining Fogel et al. with CvsIn. Appellant respectfully disagrees.

In Claim 16, as also argued above with regard to claim 1, the term **program construct** cannot be interpreted as a whole file as the Examiner asserted in the rejection of Claim 1. Therefore, in addition to not being able to interpret the term inconsistent with Appellant’s specification as the Examiner did in rejecting Claim 1, Claim 16 is also not taught by Fogel for the reasons stated in the patentability arguments of Claim 1.

That is, Fogel et al. do not teach (1) a constructor to determine, when executed by a processor, whether a construct in an edited program has been edited; (2) a construct list within the repository into which all constructs that are derived and/or related to the edited construct are listed; (3) a parser to parse the edited construct and the derived and/or related constructs, when executed by a

CA920020065US1

processor; (4) a matchmaker to determine, when executed by a processor, all the constructs that are derived and/or related to the edited construct and to enter all constructs determined to be derived and/or related to the edited construct in the construct list; and (5) an announcer to notify, when executed by a processor, any of a plurality of programmers accessing the integrated development environment that the edited construct has been edited and the constructs that are derived and/or related to the edited construct as claimed.

Thus, combining the teachings of Fogel et al. with those of CvsIn does not teach the claimed invention.

Hence, Claim 16 is patentable over the applied references.

### **Claim 17**

Claim 17 includes the limitations of Claim 16 and further includes the limitations "further comprising listing within the construct list all source code derived from the edited construct and/or any of the other constructs in the construct lists derived and/or related to the edited construct."

As mentioned above, Fogel et al. do not teach a construct list. Therefore, Fogel et al. do not teach listing within the construct list all source code derived from the edited construct and/or any of the other constructs in the construct lists derived and/or related to the edited construct as in the claimed invention.

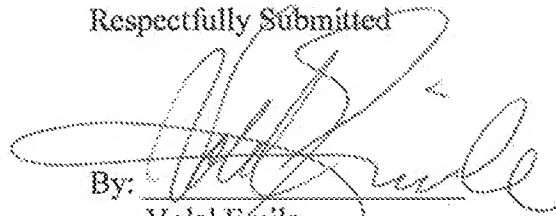
Thus, combining the teachings of Fogel et al. with those of CvsIn does not teach the claimed invention.

Hence, Claim 17 is also patentable over the applied references.

Based on the above-posed arguments, Appellant requests reversal of the rejection of the claims in the Application.

Appl. No. 10/692,115  
Appeal Brief dated 02/25/2008  
Response to Office Action of 09/25/2007

Respectfully Submitted

A handwritten signature in cursive script, appearing to read 'Volel Emile', written over a horizontal line.

By:

Volel Emile  
Attorney for Applicants  
Registration No. 39,969  
(512) 306-7969

(viii)

Claims Appendix

1. An algorithm to improve efficiency of editing source code, comprising  
  
recognizing that a source code has been edited;  
  
identifying a program construct having the edited source code;  
  
constructing a construct list of at least one other construct having derived and/or related code to the program construct;  
  
comparing the at least one other construct with the program construct having the edited source code; and  
  
if, in response to comparing the at least one other construct with the program construct, a commonality between the at least one other construct and the program construct is found to be equal to or beyond a threshold of similarity, then notifying a user responsible for the at least one other construct that the source code of the program construct has been edited.
2. The efficiency algorithm of claim 1, wherein identifying the program construct further comprises parsing tokens of the edited source code.
3. The efficiency algorithm of claim 1, wherein constructing a construct list further comprises determining that the at least one other construct is of at least a threshold size for placement in the construct list.

4. The efficiency algorithm of claim 3, further comprising parsing a sequence of tokens from each of a plurality of constructs of the at least threshold size.
5. The efficiency algorithm of claim 4, wherein comparing the at least one other construct with the program construct further comprises comparing the parsed tokens of the edited source code with parsed tokens of each of a plurality of constructs in the construct list.
6. The efficiency algorithm of claim 5, wherein comparing the parsed tokens further comprises weighting the compared tokens to establish a degree of similarity.
7. The efficiency algorithm of claim 6, further comprising summing the weights of the compared tokens to determine if the sum is equal to or beyond the threshold of similarity.
8. The efficiency algorithm of claim 1, further comprising storing the construct list.
9. The efficiency algorithm of claim 1, wherein the efficiency algorithm is a machine-implemented process in an integrated development environment.
11. A method of determining if two or more constructs in a repository of source code in an integrated development environment are related to each other, said method comprising the steps of:

identifying a first construct;

parsing N tokens of the first construct, N being a positive integer;

identifying a plurality of other constructs in the repository;

parsing M tokens of each one of the other constructs, where  $M = N$ ;

comparing the M tokens of each one of the other constructs with the N tokens of the first construct;

determining a weight for each one of the N and M tokens based on name, type, and/or representation;

summing the weights of the N and M tokens;

determining whether the sum of the weights of the M tokens meets or exceeds a threshold of similarity, the threshold of similarity being based on a percentage of the sum of the weights of the M tokens to the sum of the weights of the N tokens; and

identifying each construct whose sum of the weights of the M tokens meets or exceeds the threshold of similarity as being related to the first construct.

12. The method of claim 11, wherein the step of identifying the first construct further comprises the step of identifying whether a source code within which resides the first construct has been edited.
13. The method of claim 11, further comprising storing a pointer to each construct identified as being related to the first construct in a construct list of related constructs.

14. The method of claim 13, further comprising the step of identifying users responsible for each of the constructs in the construct list.
15. The method of claim 14, further comprising the step of offering notification to each user responsible for each one of the constructs in the construct list.
16. An integrated development environment, comprising:
  - a repository of source code into which programs in the integrated development environment are stored;
  - a constructor to determine, when executed by a processor, whether a construct in an edited program has been edited;
  - a construct list within the repository into which all constructs that are derived and/or related to the edited construct are listed;
  - a parser to parse the edited construct and the derived and/or related constructs, when executed by a processor;
  - a matchmaker to determine, when executed by a processor, all the constructs that are derived and/or related to the edited construct and to enter all constructs determined to be derived and/or related to the edited construct in the construct list; and
  - an announcer to notify, when executed by a processor, any of a plurality of programmers accessing the integrated development environment that the edited construct has been edited and the constructs that are derived and/or related to the edited construct.



17. The integrated development environment of claim 16, further comprising listing within the construct list all source code derived from the edited construct and/or any of the other constructs in the construct lists derived and/or related to the edited construct.

20. An article of manufacture, comprising a data storage medium tangibly embodying a program of machine readable instructions executable by an electronic processing apparatus to perform method steps for operating an electronic processing apparatus, said method steps comprising the steps of:

determining that a source code has been edited in an environment of computer program development;

determining if the edited source code is within a construct of a size of a particular range;

parsing the construct having the edited source code if the edited source is within a construct of the size of the particular range;

finding and parsing other constructs in the environment having a size within the particular range;

creating a construct list of other constructs in the environment having a size within the particular range;

comparing tokens of the construct having the edited source code with tokens of the other constructs in the construct list; and

determining that the construct having the edited source code is related to any one of the constructs in the construct list if the tokens of any one of the constructs in the construct list equal the tokens of the construct having the edited source code.

21. The article of manufacture of claim 20, further comprising weighting the compared tokens based on value, type, and/or representation.
22. The article of manufacture of claim 20 wherein said method steps further comprises the step of notifying a user responsible for of any one of the constructs in the construct list that is determined to be related to the construct having the edited source code that the source code of the construct has been edited.
23. The article of manufacture of claim 20, further comprising at least one construct list of related and/or derived constructs within the integrated development environment.

(ix)

Evidence Appendix

None.

Appl. No. 10/692,115  
Appeal Brief dated 02/25/2008  
Response to Office Action of 09/25/2007

(x)

Related Proceedings Appendix

None.